Personalization of Context-dependent Applications through Trigger-Action Rules

GIUSEPPE GHIANI, CNR-ISTI, HIIS Laboratory
MARCO MANCA, CNR-ISTI, HIIS Laboratory
FABIO PATERNÒ, CNR-ISTI, HIIS Laboratory
CARMEN SANTORO, CNR-ISTI, HIIS Laboratory

Our life is characterized by the presence of a multitude of interactive devices and smart objects exploited for disparate goals in different contexts of use. Thus, it is impossible for application developers to predict at design time the devices and objects users will exploit, how they will be arranged, and in which situations and for which objectives they will be used. For such reasons, it is important to make end users able to easily and autonomously personalise the behaviour of their Internet of Things applications, so that they can better comply with their specific expectations. In this paper we present a method and a set of tools that allow end users without programming experience to customize the context-dependent behaviour of their Web applications through the specification of trigger-action rules. The environment is able to support end-user specification of more flexible behaviour than what can be done with existing commercial tools, and it also includes an underlying infrastructure able to detect the possible contextual changes in order to achieve the desired behaviour. The resulting set of tools is able to support the dynamic creation and execution of personalized application versions more suitable for users' needs in specific contexts of use. Thus, it represents a contribution to obtaining low threshold / high ceiling environments. We also report on an example application in the home automation domain, and a user study that has provided useful positive feedback.

• Human-centered computing ~ User interface programming • Human-centered computing ~ Ubiquitous and mobile computing systems and tools

Additional Key Words and Phrases: End-User Development, Internet of Things, Trigger-Action Programming

1. INTRODUCTION

End-User Development (EUD) aims to put the applications development in the hands of the people who are most familiar with the actual needs to be met (e.g. domain experts). However, the design and development of flexible software able to match the many possible user needs and provide high quality user experience is still a major open issue. In addition, the explosion of mobile technology and the Internet of Things (IoT) have further increased the wide variability and heterogeneity of the possible contexts of use, and have exponentially increased the number of dynamic events that can occur in them. Thus, the complete behaviour of context-dependent applications cannot be hardcoded by developers at design time because they cannot foresee all the possible and even unpredictable situations the applications would encounter during use and whether the produced results will be actually meaningful to end users.
Moreover, existing software development cycles are still too slow to quickly respond to rapidly changing user needs of variegated categories of users, and professional

Author's addresses: CNR-ISTI, Via Moruzzi 1, 56124 Pisa, Italy.

DOI:http://dx.doi.org/10.1145/0000000.0000000

developers often lack the needed domain knowledge to address such requirements. We believe that a viable way to make context-dependent applications comply with users' expectations is to have users themselves program the specific dynamic behaviour they need. This means making end users more active in the development process, and enabling them to directly shape the applications provided by professional developers to quickly respond to and address their unique, individual and often transient needs. This is particularly true for Internet of Things applications [Atzori et al. 2010] in which a variety of technologies (including RFID tags, sensors, actuators, etc.) can be involved. In the Internet of Things vision, 'smart' physical objects are networked together, able to interact and communicate with each other, with human beings and/or with the environment to exchange data and information 'sensed' about the environment, reacting autonomously to events in the real world, and influencing it by running processes that trigger actions and perform services. In this scenario IoT applications need to address extremely *contextualized* user needs. Indeed, one of the primary concerns of developers of IoT-based applications is managing the heterogeneity of contexts of use in terms of devices, sensors, actuators, and services involved in IoT-related domains. Such challenges can be addressed by solutions able to properly collect, model and interpret contextual data in order to provide significant added value to applications.

In order to simplify the development of such applications, an important role is played by the availability of a distributed context management middleware separated from applications. Its goal is to hide the heterogeneity of IoT hardware and provide applications with context-management services related to context acquisition, modelling, storing and processing, so end users do not need to directly manage them and can focus on designing when and how the behaviour of their applications should adapt. In addition, this intermediate middleware for managing contextual events provides end users with meaningful logical descriptions of events and conditions able to abstract out from low-level details in the raw data provided by the available variety of sensors and devices. In this way users can more easily focus only on the dynamic aspects of the context of use and think about how to personalize their applications accordingly. To this regard, adopting a *trigger-action paradigm* represents a promising approach because of its compact and intuitive structure, which directly connects the dynamic events and/or conditions with the expected reactions without requiring the use of complex programming structures.

Various IoT-based application domains can benefit from such trigger-action paradigm. For example, a retail manager can define rules for sending personalized advertisements to customers' smartphones based on their movements and interactions with products in the shop; a caregiver can define rules for sending elderly patients personalized reminders for promoting healthy behaviour: for instance, by sending an alert through their favourite device (e.g. the TV) to encourage them to do more exercise, based on the activities detected; a construction manager can define rules to trigger safety warnings to workers on their smartwatches based on the dynamically detected presence of some potentially dangerous equipment; home dwellers can set up rules for better controlling tasks in their home based on their own specific preferences, routines, available equipment and current context, for instance if it is a week day, between 9 am and 5 pm, and the motion detector camera detects some movement, the application should flash red lights on the user's smartphone, also sending user a message with an image from the

home webcam screen capture. In addition, since the development of IoT applications requires a variety of skills (e.g. domain, low-level hardware, network knowledge), the use of meaningful and intuitive EUD concepts, metaphors, vocabularies and notations should allow the different stakeholders not only to comprehensively handle the system, but also to easily communicate ideas and concepts through a common reference.

EUD based on trigger-action rules is expected to allow users to do more (and more easily) with their existing devices and things by softening the boundaries between "end users" and "professional developers" as well as between design done before use and software adaptation done at runtime. By specifying their personalization rules, users should be able to get better support and more satisfaction in the use of their context-dependent IoT-based applications. This type of solution can thus contribute to creating technological infrastructures that can successfully establish their usage in practise [Pipek and Wulf, 2009] if it is able to address the specific challenges for obtaining low threshold / high ceiling environments [Myers et al., 2000].

The main goal of this work is to identify a method that can properly support the design and development process of EUD-enabled context-aware Web IoT-based applications. The method is supported by an authoring tool for specifying trigger-action rules and an architecture based on a context manager that is able to activate, interpret and apply such rules to the applications considered. We have focused on the Web because of its pervasiveness in terms of supporting devices (it can be seamlessly accessed through a variety of devices), and thus it can best support and leverage context-dependent scenarios. In this area some commercial tools have started to appear: IFTTT[1] is a common tool that allows people without programming experience to create simple applications such as “If I arrive home then turn lights on”. Besides being able to express rules that involve the hosting device, IFTTT is also able to communicate with widely used Web services. However, one of the main disadvantages of this solution is that it is rather limited in terms of expressiveness since it does not allow users to create more structured rules, i.e. those combining multiple events and actions [Ur et al. 2014].

The proposed environment allows end users to customize the original application by specifying trigger-action rules that indicate the desired specific application behaviour for the target contexts of use. Within such rules triggers are associated with dynamic changes that can occur in the targeted context of use (in terms of users, devices, things, environments, etc.). The actions are performed when a trigger is verified, and indicate changes to carry out in order to achieve the expected adaptation, i.e. new functionalities to activate or modifications in the application user interface and/or logic. The environment is decoupled from the application that has to be adapted and, as such, is generic and can be applied to different domains. Thus, to summarize, the main contributions of the presented solution are:

- An authoring environment for enabling end users to specify expressive trigger-action rules (with various possible compositions of triggers and actions, and with a clear distinction between events and conditions that can define triggers) in a way understandable by end users;

[1] https://ifttt.com/

- The integration of the authoring environment with a context manager middleware able to detect the events generated by the various sensors and devices. This integration allows the rule editor to show the rules that can be actually executed and immediately apply them;
- A generalisation of our method that enables it to be applied with minimal effort in multiple distinct domains (e.g. home, smart retail, and ambient-assisted living). For the most part, customization for a specific application domain requires only some refinements in the context model, which determines the behaviour of the context manager middleware, and consequently the detectable triggers.
- An example application of the method and tools for application personalization through trigger-action rules in the home automation domain. Smart homes consist of networked devices and things exposing well-defined programming interfaces allowing the creation of applications through which users control their smart environments. However, currently, such applications offer just the specific functionalities that device manufacturers or software vendors planned for them while we provide users with the possibility of more flexible and general personalization support.

The article is structured in the following manner: after discussing some related work (Section 2), we introduce the requirements, the basic concepts and the design method followed (Section 3), then we present the overall underlying architecture of the environment (Section 4) followed by an illustration of the TARE (Trigger-Action Rule Editor) environment (Section 5) . We also show an example application in the smart home domain (Section 6), and report on a user test carried out to gather user feedback on the authoring tool (Section 7). Finally, we provide some conclusions and directions for future work.

2. RELATED WORK

Our work draws from research on end-user development for Internet of Things applications and frameworks and tools for context-dependent applications.

2.1 End-user development for Internet of Things Applications

[Atzori et al. 2010] surveys the Internet of Things area mainly from a technical perspective (e.g., by discussing the pros and cons of enabling technologies such as RFID and TCP), but also mentions the benefits of combining sensors and actuators with personalization techniques: managing home appliances based on user preferences and dynamic contextual factors can improve comfort, safety and energy efficiency. A concrete example of such approaches is given in [Buckl et al. 2009], which describes a home prototype to minimize electricity consumption expenses by automatically turning on/off devices and switching between public grid and a domestic UPS. However, its Web service composition mechanism does not seem suitable for the end user. Some work to address such issues in the EUD perspective has started to be put forward. One of the first proposals was iCAP [Dey et al. 2006], which introduced the possibility to create if-then rules and to support personalization of dynamic access to home appliances. We aim to provide an environment able to support intuitive editing of a broader set of rules in terms of possible trigger and action types, and with additional possibilities, such as rule reuse and sharing. In this way we can provide more flexible and general support also with respect to smart

home hubs such as Wink Hub[2], which allows users to control home appliances through what they call "robots", which trigger actions depending on aspects such as location or movement. However, such smart home hubs only work for a predefined set of appliances and conditions, and do not allow users to freely compose multiple triggers. SPOK is an architecture aiming to support non-expert users to program their smart environments [Coutaz and Crowley 2015], [Coutaz et al. 2014]. It is structured into two abstraction levels, the so-called 'Core World' including devices, sensors, and cloud services, while the 'Extended World' defines functionality relevant to the application domain. To build programs, end users access a Web-based editor with pseudo-natural syntax, and an interpreter allows them to test the program. The EUD environment has then been further extended in a new version, called AppsGate, which has been deployed in real domestic environments [Coutaz and Crowley, 2016]. Our environment aims to be as general as possible, supporting the possibility to specify a wider set of complex triggers and actions that can be customized for various application domains. Drey and Consel [2012] have proposed Pantagruel, a visual editor for end user development of home automation rules combining sensors and actuators. The environment supports the definition of complex conditions, however the rules considered in the reported user test were relatively simple. We are interested in enabling the average user to model flexible behaviours by defining triggers involving various contextual parameters.

Perera et al. [2015] studied how a natural language approach can support the definition of policies to manage the domestic environment. They only considered the "sticky note" technique for defining the tasks requiring information exchange between IoT appliances and services. The findings revealed that the average number of words per note was relatively small. Overall, the initial hypotheses were confirmed: people in general adjust their language depending on the type of addressee (human vs. machine), and their technical background affects the way users communicate with machines. We took into account such findings by minimizing the number of words for describing user-edited rules in natural language. Our work however goes beyond the simple creation of rule descriptions, as we enable users to create rules that can be actually executed in smart environments.

A recent study [Lucci and Paternò 2014] about expressiveness and usability of mobile Android apps for allowing users to configure dynamic behaviour found a lack of consistent terminology: each environment provides different names for similar concepts, which does not help users to immediately understand them. The most expressive environment (Tasker[3]) was also the one that was found most difficult to use (highest performance time, error numbers, and unsuccessful performance numbers). With the increasing number of categories for grouping the relevant concepts, there is also an increasing risk of misunderstandings unless familiar classifications, icons and metaphors are proposed to represent and manage such concepts. Since there are many possible elements, they should be structured according to intuitive logical categories that match the mental representation of mobile users.

[2] http://www.wink.com/products/wink-hub/
[3] https://play.google.com/store/apps/details?id=net.dinglisch.android.taskerm&hl=en

The availability of mobile tools to perform real time check of the configuration of on-site visual interactive systems is deemed essential in [Kubitza et al. 2015] to accelerate the so called "change and re-try cycles". An example tool for configuring smart environments is described in [Kubitza and Schmidt 2015]. It aims to facilitate physical prototyping by hiding the platform/communication/device complexity that arises when many different technologies are combined together. The tool is structured so as to have a separation of the management of devices, events and rules, and mainly targets programmers since the rules are based on JavaScript, while our authoring tool distinguishes between rule triggers and actions, and we target the average end-user who typically is not a professional programmer able to manage JavaScript.

IFTTT is a popular environment that allows users to easily connect existing applications in such a way that if something happens in one, then some effect can be generated (for example a functionality is activated) in a kind of trigger-action programming. One of its distinguishing features is that, besides being able to express recipes that concern the hosting device, it communicates with widely used Web services, thus allowing the automatic execution of functions related to the internal state of apps such as Facebook, Instagram, Ebay, YouTube and others. A recent study [Ur et al. 2014] found that trigger-action programming can express most desired behaviours in order to customize smart home devices. They also found that inexperienced users can quickly learn to create programs containing multiple triggers or actions obtained by extending the IFTTT language, which has limited possibilities, since it only supports applications with one trigger and one action. This shows that this approach seems suitable to support EUD of context-dependent applications, but needs to be improved in order to allow users to express the various desired combinations of events and corresponding actions. As for the triggers managed, IFTTT provides predefined lists of triggers associated to the services that have been connected to this environment, and the development process supported is sequential. When compared to our approach, IFTTT seems to exploit a rule-based metaphor of a similar level of intuitiveness. However, one of the main disadvantages of IFTTT is its limited expressivity since it only allows selection of a single trigger and a single action per rule from predefined lists. A more detailed comparison of IFTTT with our approach will be provided after presenting our Trigger-Action Rule Editor (see Section 5).

Huang and Cakmak [2015] discuss current trigger-action programming trends and issues. In particular, they found that the distinction between relevant concepts is source of problems, since users can have difficulties interpreting the difference between events and conditions or between the possible types of actions (for example extended actions, which reverts back to the original state after some time automatically and sustained actions, which do not revert to the original state automatically). Misunderstandings can cause undesired behaviours (e.g. unlocking doors at the wrong time or cause unintended energy waste). When designing our authoring tool, we have taken into account the requirements emerging from this study, for example allowing users to differentiate between event triggers (that hold only when a contextual change occurs) and condition triggers (that hold whenever a condition is true), while other approaches such as IFTTT do not provide support in this respect.

In general, there are two main approaches to application composition [Davidyuk et al. 2015]. In the automated composition, user intervention is minimal since the system automatically configures and provides most of the functionalities. In the interactive composition, the user has a high degree of control and can freely compose the final application. For interactive application composition various metaphors have been proposed, such as pipeline, jigsaw puzzle, and join-the-dots. In our work we focus on customization of existing applications and we allow users to interactively compose their contextual rules while providing a structured representation of the relevant concepts, which facilitate their composition work. A different approach is illustrated in [Desolda et al. 2015] for mashing up smart things (sensors, actuators). In mashup approaches the basic point is to facilitate new compositions amongst existing components, while we find more flexible to add incrementally new contextual rules for modifying the original behaviour of the interactive application.

2.2 Frameworks and Tools for Context-dependent Applications

The Context Toolkit [Salber et al. 1999] was among the earliest supports for developing context-enabled applications by providing a library to facilitate integration with sensors. It initially considered a limited set of events and led to meld the context awareness code with the application. More recently, the Context Toolkit has been augmented with support to facilitate development and debugging of context-dependent applications [Dey and Newberger 2009]. Programming abstractions, called Situations, expose an API supporting both developers and designers to provide application adaptivity without coping with low-level code. A further extension of the toolkit was devoted to improving the intelligibility of the context-dependent applications, thus better supporting the end-user in understanding/foreseeing the application behaviour. Our work has a different objective, i.e. to investigate what level of complexity end-users can achieve in customizing context-dependent applications by themselves. However, such intelligibility aspects are also relevant for our authoring tool, since it should allow non-programmers to define valid and semantically correct rules.

A context-aware system has been proposed by Anh and Kim [Ahn and Kim 2006]. In their environment the network nodes cooperate in a distributed manner, and the way the system should react to context changes is encoded in Context Descriptors, which are similar to our adaptation rules (i.e. they bind events and conditions with actions). However, the authors do not investigate the usability issues of creating Context Descriptors by end users. We are instead interested in studying the implications of rule editing from the end user viewpoint, also when modelling multiple triggers and/or actions.

Hu et al. [2008] report three main approaches for the development of context-dependent applications: no application-level context model, implicit context model, explicit context model. We have chosen the third approach to context management, i.e. the applications share a context management infrastructure, which is able to populate the state of the context model by means of external sources of contextual information. Van Bunningen et al. [2005] classified the ways of categorizing context into operational and conceptual. In the former, context information is categorized according to the way it is collected and modelled; in the latter, it is categorized based on conceptual relationships. With respect to such classification, our context

management solution acts on a conceptual level, since classification and access are performed according to four main dimensions (user, environment, technology, social).

SOCAM [Gu et al. 2005] is an architecture for rapid prototyping of context-dependent services that relies on two-level context ontologies. The upper ontology defines general concepts, while the domain-specific ontologies are low-level ontologies able to capture domain oriented contexts (e.g. smart home, vehicle, etc.). Even if we do not explicitly define ontologies, we also adopted a two-level approach on our context management solution: we have indeed defined a generic context model that describes classes of objects of the general world. We then define a specific context model for each application domain. The domain-specific model exploits a subset of the information classes defined by general model and may redefine terms and relationships among them. The issue of defining valid and complete rules for context reasoning is tackled by Guan et al. [Guan et al. 2007]. They refer to "context relationship" as the knowledge of which low-level context features to choose for a high level reasoning, which depends on the user's domain theory accuracy. The method proposed automatically filters out useless contextual parameters in order to refine human-defined rules. Our approach to rule definition is instead human-driven and, to this end, we support domain-specific context models in order to improve the trade-off between completeness/expressivity and intuitiveness/simplicity.

Dax et al. [Dax et al. 2015] proposed FRAMES, a framework for supporting both researchers in defining and adjusting their studies, and users to respond to open and closed questions triggered by certain (complex) events. Our aim is to provide general support for end users in composing triggers and actions for personalizing Web applications. Mayer et al. [2014] have put forward a proposal for model-based generation of Internet of Things applications based on taxonomy of abstract sensing and actuation primitives. We exploit a context model in order to support users without programming experience to select the relevant triggers for the rule that should customize the dynamic behaviour of their applications. In [Ghiani et al., 2015] a proposal for the development of context-dependent cross-device user interfaces based on trigger/action rules has been put forward. Unfortunately, that tool was usable only by professional developers and when it was shown to domain experts without programming experience they raised various issues and had difficulties in understanding the proposed concepts and how to manipulate them.

Our solution draws inspiration from such previous work, and provides a novel methodological contribution showing how meta-design can be obtained in context-dependent applications by involving domain experts and end users. This is obtained through an authoring environment able to support end user specification of more flexible personalization rules, which can be specified by end users without any programming experience, and a context manager integrated with the authoring environment, which support direct and continuous access to the sensors and devices in the contexts of use considered for their execution.
This is an important contribution as it aims to fill the current gap in effectively supporting end-user development of context-aware IoT applications, which prevents to exploit the IoT to its maximum potential – i.e. where end-users can take control and co-create solutions that fit their own needs and context. A number of potential advantages associated with our solution can be identified and summarised as it follows: i)reduction of the time to market of IoT-based applications: with our solution

end users will be directly involved in the development of context-dependent IoT-based applications, thus it will be better and more quickly ensured that the system will comply with what is actually expected by users; ii) a productivity increase in all aspects of software life-cycle: from quicker and more precise requirement analysis (due to the direct involvement of end users), to increased productivity in developing customised software meeting user's expectations, to easy support for software maintenance (due to the capability of the approach to support software evolution and adaptation); iii) ability to meet software quality levels required by a fast growing number of software-enabled products and services: the approach proposed aims to provide users with means to empower them to customise their software applications in a context-dependent manner; iv)increased reuse of design/development artefacts in the development of new software. Our trigger-action rule approach will provide an easy manner to share and reuse code between different applications and contexts of use, so reducing the costs of developing new software from scratch and opening new capabilities for companies that cannot afford the costs of either traditional software development. Furthermore, end users can find the trigger-action rules useful to apply even in different contexts, or to share them with other users, or to use them as starting point for creating new context-dependent rules.

3. REQUIREMENTS AND DESIGN METHOD

As mentioned before, our method is aimed at allowing end users without programming experience to customize the context-dependent behaviour of their Web applications. Such personalisations are expressed by specifying relevant trigger-action rules. In this section we analyse the main design aspects of our solution as well as its main conceptual elements.

First (Section 3.1), we discuss the challenges and requirements associated with the customisation of context-dependent IoT-based applications. Then, we describe the *context model* (Section 3.2) that is exploited in the tools in order to allow end users to specify relevant contextual events and conditions (the triggers) in a logical, high level manner, i.e. by abstracting out the peculiarities of heterogeneous hardware and software. Similarly, we have also classified the possible *actions* that can be included within the personalization rules (see Section 3.3). Finally, in Section 3.4, we better detail how the design and development process associated with the proposed tools can be structured into a number of phases that foresee the contribution of all the different stakeholders of IoT ecosystems with their own specific skills and knowledge.

3.1 Requirements

In order to identify the requirements that our environment should satisfy, we carried out a literature review of relevant work in the area, and considered previous experiences in developing context-dependent applications in various projects that dealt with different IoT application domains (smart retail, elderly assistance, warehouse picking, museum guides, construction sites). Indeed, the requirements elicitation for context-dependent applications involved various stakeholders in heterogeneous sectors and resulted in a number of events having different levels of user involvement, and different formal/informal structuring. Below we provide some details on how it was conducted in the various domains considered.

Smart Retail. In the retail domain, we mainly gathered requirements and user feedback by participating to two fairs. One was the main Italian fair dedicated to ICT, where we had an exhibition stand to present the main capabilities of a first EUD

prototype adapted for the retail domain. While demonstrating the prototype to interested visitors, we had the opportunity to informally discuss it with them and get some feedback about further requirements, possible design improvements and general users' attitudes toward the proposed solution. We had overall 15 users, and the vast majority consisted of skilled software developers, who mainly focused on the technological characteristics of the solution. The same prototype was also shown at another fair, more dedicated to retail specialists. In this case the discussion was guided by a questionnaire, which helped to focus on more concrete aspects, and which was divided into four sections: user-aware services, environment-aware services, technology-aware services and social-aware services. A further section aimed at identifying possible rules potentially useful in this domain. Once again about 15 users visited our stand. We had representatives from chain stores, service providers for the retail sector, large scale retail distribution stores, and providers of software applications for the retail sector. Our goal was to find out which services could be interesting and useful for their customers, how to provide them with added value, and discuss the potentialities that contextual information offer to promote purchase-oriented actions.

Construction sites. Another domain which was also analysed was the one dedicated to construction sites. We had discussions with a person working for a multinational company operating in construction projects, and especially concerned with safety of workers in building sites. With the help of this expert, we identified relevant requirements by means of formulating relevant scenarios that would benefit from the proposed solution. This domain allowed us to explore the potentiality of our solution in peculiar, heterogeneous outdoor environments, since construction is a complex sector that involves different stakeholders. Moreover, during the construction process dynamic events are also quite common (e.g. related to workers, heavy-machinery, trucks, cranes, as well as weather, natural hazards, ground conditions and other external agents). In the end, three main scenarios were identified, associated with potentially risk factors in construction sites: one dealt with traffic-related issues (due to e.g. heavy machinery, trucks, cars); another one was connected to the risks associated with suspended loads; the third one was associated with gasses dilution.

Museums. In the museum sector, we had interactive discussions with two curators of a museum. Both had several years' experience with using software applications, however they had never used any tool for customising their applications (e.g. dynamically managing presentation of exhibits or artworks whose location within the museum often changes). In this case a questionnaire was used to drive the discussion in order to gather their opinions/suggestions/attitudes about the concerned environment. They were interested in scenarios such as a museum manager that could set up a rule triggering a more detailed description of an artefact in a mobile guide if the system detects that the visitor is lingering in front of it beyond a specific amount of time.

Warehouse picking. Warehouse picking is part of logistics processes often found in retail and manufacturing industries. Warehouses store the goods and products incoming from suppliers until they are collected and shipped to the stores or customers. Here we focused on the activity of picking items from a shelf, collecting them in containers and transporting them to certain locations. Technological solutions have gained importance in the task of supporting the picker, especially when there are some contextual events that can affect the efficiency of the process. In this case they were interested in scenarios such as a warehouse operator that could

set up a rule showing an alternative path to reach an item when the system detects that the current path is blocked by other workers.
During the project, we had several opportunities to discuss with domain experts about the benefit of dynamically adapting the applications used by the pickers according to dynamic contextual events (e.g. congestion in the pick lanes, need to manage fragile objects). In this case, requirements elicitation was done using different means: by doing interviews with both developers and consumers, by identifying specific project-wide use cases, by analysing the current market, and through a dedicated focus group in which representatives of all the project partners participated (around ten persons), ranging from software companies, to research organisations, universities, standardisation bodies and service providers.
Remote assistance for Elderly. We also focused on elderly remote assistance. For the requirement analysis we identified three different classes of users: elderly, informal caregivers, and formal caregivers, and for each category separate requirements elicitation was conducted aimed to understand usual practices, and possible relevant customisations. For the older adults the emails were sent to all the contacts included in a database of a foundation operating as a representative body for mature people. The filling out was completely anonymous and on a voluntary basis. In the end, a sample of 71 completed questionnaires were collected. In order to gather requirements about informal caregivers (e.g. partner of the elderly), we used two different techniques: Personas method and an online survey. As for personas, we identified three different senior beneficiaries (personas) of the solution. In addition, we also sent out a questionnaire to the relatives' contacts provided by the foundation. The filling out was completely anonymous and on a voluntary basis. In the end, a sample of 13 completed questionnaires were collected. In the case of health professionals, we were looking for a representation of what the current practices employed by health professionals and therapists are, as well as these professionals' opinions on their limitations, and expectations created by future technological solutions, thus we opted for a qualitative approach. Two focus groups were conducted with members of a rehabilitation hospital. In such focus groups the context and overall goals of the project were first introduced, then we discussed how, currently, older adults receive care and support from the municipality services, and finally, we debated how some of the concepts the solution wishes to introduce could be adopted and improve current practices.

It is worth pointing out that across such events we followed an evolutionary approach since requirements were more vague at the beginning, and gain more precision while the discussion with stakeholders proceeded and the acquisition of the knowledge of the domain by designers improved and become more accurate. In addition, with stakeholders, the issue of prioritisation of requirements was more focused at a domain-dependent level. For instance, while discussing with retailers it came frequently out that they were more concerned with requirements allowing to satisfying certain business goals. This is very different from e.g. home settings where users are often more concerned with automating common routine tasks in an easy manner, as well as work environments in which one of the primary goals would be the efficiency and performance of workers.

In the end, a number of requirements were identified, summarised in Table 1 and discussed in detail in the following.

Table 1: An overview of the identified requirements

R1	Need for a meta-design approach that integrates contextual IoT information
R2	Possibility of combining multiple triggers and/or multiple actions
R3	Distinction between events and state conditions
R4	Enhance visual EUD languages with natural language support
R5	General vocabulary for describing context of use customizable for different application domains
R6	Possibility to reuse the rules
R7	Provide flexible yet simple tools
R8	Means for rule conflict resolution
R9	Means for simulating and debugging rules

R1. Need for a meta-design approach that integrates contextual IoT information. There is a need for a participatory, collaborative process in which end users, domain experts and developers contribute with their different expertise at various stages of the process in co-creating the software artefacts to obtain meaningful results in real contexts and address concrete end user needs [Fischer et al., 2004]. IoT applications introduce one further aspect to consider in meta-design approaches: how to design when and how an application reacts to events generated by the various sensors and devices. Thus, an effective solution should support suitable integration of the meta-design process with a Context Manager providing a unified and homogenous view of context information coming from different sources in order to allow people with different backgrounds and programming experience to easily manage the contextual events generated by the sensors and devices. For instance, discussions with smart retail experts revealed the need to have various context-based customisation steps even within the same retailer, so as to accommodate the needs of both small shops and large stores having the same brand. In these cases, although the manipulated application concepts are the same (e.g. products, shelves) the specific contextual settings can radically differ since the same application could be required to work in different concrete contexts of use (in terms of environments and available technologies) referring to the same retailer.

R2. Possibility of combining multiple triggers and/or multiple actions. Even if previous studies indicated that users do not tend to create particularly complex rules [Dey et al. 2006], it is still important not to limit them to formulating rules composed of only one trigger and one action as in IFTTT. A study has shown that such possibility should not create particular problems to users [Ur et al. 2014]. The utility of such requirement came out, for example, during the focus groups with members of a rehabilitation hospital, in order to better manage the complexity of conditions in which rules are typically triggered in that domain. Indeed, in elderly assistance the rules are often triggered by checking specific events in the context surrounding the elderly, which should be cross-checked with conditions regarding the person's health-related status (which in turn can involve multiple aspects, depending on their diseases).

R3. Distinction between events and state conditions. Recent user studies [Huang and Cakmak 2015] show that current environments sometimes create confusion in end users concerning the difference between these two concepts: events are associated with when a state changes, while conditions refer to persistent states, i.e. those that

can last for longer periods of time. A factor potentially contributing to this confusion is that these two concepts are often related to each other (the condition "Tom is in the kitchen" is strictly related to the preceding event of "Tom entering the kitchen"). Thus, it is useful to make the distinction between such two concepts clear.

R4. Enhance visual EUD languages with natural language support. The possibility to support multimodal user interfaces capable of receiving both some subset of natural language and visual language inputs in a complementary manner is highlighted as still untapped [Perera et al., 2015]. Indeed, systems such as IFTTT, Atooma[4], and Locale[5], are mainly based on only visual iconic languages (in which events and actions are represented through icons), and therefore they do not support multimodal ways to express the desired behaviour. This requirement was identified during our discussion with construction site and warehouse experts, who highlighted the need to reach the user in a natural and intuitive manner, which is especially important for people who are involved in work activities and thus share their attention between the EUD tool and job tasks.

R5. General vocabulary for describing context of use customizable for different application domains. Previous environments, such as [Dey et al. 2006] utilize different types of rules to adapt applications in a context-dependent manner, in the case of iCAP they are: simple if-then rules, (personal, spatial and temporal) relationship-based actions. Such rules have been identified depending on the specific application concepts to handle. Thus, there is a need for a more general common vocabulary for describing the possible contexts of use, structured and conceptualised in such a way to guide users in the selection of the relevant concepts to include in their rules when indicating the triggers and the actions. Such vocabulary can then be specialised for the various possible target application domains.

R6. Possibility to reuse the rules. The Trigger-Action paradigm can be applied beneficially since it presents solutions for recurring problems associated with dynamically processing context information and proactively reacting upon context changes. As such, the end user can find it useful to apply them even in different contexts or to share them with other users or to use them as starting point for creating new rules. Therefore, it is important to provide users with tools enabling them to save the rules for possible future use in a different context/domain. This requirement has been found useful in a recent study [Tetteroo et al., 2015] on EUD in rehabilitation clinics, in which the end users were therapists needing to often combine various devices to provide patients with the best support for their exercises. During the study, the users' tendency to reuse existing artefacts (exercises previously created by colleagues) was assessed and the main motivation was the urgency of making an exercise available for a patient. This study indicates the importance of quick and effective mechanisms for reusing previously defined artefacts as well. In our experience, the reuse mechanism was found useful in all the work settings in which efficiency is important (e.g. warehouses and retail).

R7. Provide flexible yet simple tools. A previous study [Lucci and Paternò 2014] highlights that tools in this area should facilitate the understanding of the event / condition / action paradigm, and the search and use of the elements of interest. Thus,

[4] https://play.google.com/store/apps/details?id=com.atooma&hl=en
[5] https://play.google.com/store/apps/details?id=com.twofortyfouram.locale&hl=en

a usable design should be able to graphically represent the cause / effect mechanism without imposing any temporal constraint regarding which to specify first. For example, the elements of interest should be selectable from lists providing an appropriate number of elements in order to avoid user difficulties in identifying the desired elements. In addition, in order to facilitate users, it will also be critical to provide them with easy-to-learn tools (the interface should give users immediate confidence that they can succeed), and exploiting the relevant concepts and choosing the appropriate level of abstraction for such concepts. However, we should note that this requirement can include some difficult trade-offs, since in several cases increasing the expressiveness of the tool can affect the perceived simplicity by the user, thus, expressiveness is obtained at the expense of usability.

R8. Means for rule conflict resolution. In rule-based environments it is possible to have potential conflicts in the effects that the rules can have. In this case conflict resolution strategies should be put in place able to identify potential conflicts and help in finding possible solutions, even by involving users in this process. A possible conflict resolution support is to associate rules with specific priorities, i.e. an integer determining which rule should be executed before the others, as it happens in our solution. In iCAP, [Dey et al. 2006] two possible levels were considered: potential conflicts at design time and actual conflicts detected at runtime. At design time, when rules are saved, it checks whether any of the saved rules could potentially conflict and, if any conflict occurs, the concerned rules are highlighted to the user to resolve the conflict or ignore it. If rules conflict at runtime, iCAP, by default, executes the rule most recently updated.

R9. Means for simulating and debugging rules. A key point is how people can test the rules and possibly assess whether they result in the expected behaviour, e.g. they really activate the desired appliance. Here we mainly focus on the semantical correctness of the rules, i.e. when they are syntactically corrected but do not behave in the expected manner. The need to have correct rules came out as especially important in sectors where incorrect behaviour of applications or actuators can eventually have safety-critical consequences (e.g. the elderly assistance domain as well as the construction site domain). A way to reduce the likelihood of errors in the specification of rules is to allow users to *simulate* the conditions and events that can trigger a rule and the effects that they will bring about. An example of this approach can be found in [Coutaz and Crowley, 2015] in which a program interpreter and a clock simulator was developed to test program execution in "simulated time". Alternatively, rules could be actually applied and executed in the current context of use. In our solution we support both possibilities. In this way it is also possible to receive information helpful for finding the causes of the undesired behaviours detected and fixing them. In addition, it is worth noting that debugging can be a hard task even for trained programmers. For non-professional end users it becomes especially difficult because, as noted by [Coutaz and Crowley, 2016], most EUD environments do not include debugging aids for such users.

3.2 Context Model for Trigger classification

Developing context-dependent applications involves defining the relevant contextual triggers and the corresponding actions. In order to customize context-dependent applications through triggers and actions, we have designed a context model whose

structure can aid users in specifying their rules. In particular, the context model indicates the main aspects that can vary and thereby generate events corresponding to the triggers of dynamic customization rules.

The context model is structured along four main dimensions (*user*, *environment*, *technology*, *social relationships*) with the aim to describe the relevant aspects that can affect interactive context-dependent applications. In order to identify such dimensions we have considered previous work and our experiences in designing context-dependent applications. Perera et al. [2015], in their literature survey on context awareness and the Internet of Things, propose a classification of previous work by indicating various context types (User, Computing, Physical, Historical, etc.). In general, each work in their survey considered more than one context type, however some context types were addressed more often than others: "User", "Computing System", "Physical Environment", "Time" were the most commonly addressed. We go beyond this view since we structure the 'context' concept through four dimensions – user, technology, environment, and social aspects– which all can contribute to specify a particular contextual situation. Not only our dimensions are able to cover the various context types that [Perera et al. 2015] indicated, but we also consider social aspects, which have recently been emerging as important in some context-dependent applications. In the following, we detail the refinement of each contextual dimension.

The **user** dimension includes *personal data*, *physical and mental state*, *position and activity*, *personal social connections*. *Personal data* concern some static information about the user (name, age, gender, knowledge/education) and preferences, further refined into language, leisure, sleeping, eating. *Physical and mental state* concern data associated with disabilities (e.g. blindness, deafness, motor), diseases (e.g. hypertension, diabetes), emotions (e.g. anxiety, boredom, fun), cognitive aspects (e.g. attention, meditation), and physiological data (e.g. heart pulse, blood pressure, blinking). In *position and activity* the former can be specified in absolute or relative terms: relative position can be expressed in terms of proximity type (e.g. in_front_of | beside | below | …) and point of interest (e.g. device| thing| environment), with also the optional possibility of specifying their relative distance. Activity is further subdivided into behaviour and goal, the former concerning postures or movements types (e.g. standing, sitting, moving, lying), the latter, name and type of goal (e.g. physical, interactive, cognitive). *Social connections* refer to the users' social relationships. For each of them it is possible to indicate: contact name and the type(s) of relationships (note that a user can have multiple relationships with the same person, e.g. a person can be at the same time a relative but also a neighbour).

With **technology** we consider information related to any relevant technology available, e.g. devices, smart things/ appliances, network connectivity. In particular, technology is further subdivided into devices, which has a number of attributes e.g. name, type (e.g. tablet, TV, smartphone), owner, position, battery level; things/appliances, which has some attributes (such as name, type, position, state, power consumption); connectivity, which describes the different types of network capabilities of the context (e.g. Wi-Fi, infrared, Bluetooth).

The **environment** dimension is further conceptualised into characteristics, which provide general information on the surroundings. It can include information concerning: type (e.g. indoor/outdoor); name; spatial aspects, (e.g. size, shape,

position), ambient conditions (which is in turn further divided into temperature, light, noise, humidity, motion, presence, time, weather); things/appliances, which refer to those that are in the environment considered; and ambient conditions (temperature, light, noise).

The last dimension concerns **social relationships**. It includes the following: *type of network*: for instance if the network is virtual or physical; *type of relationship*: the relationship shared by the members of the network (e.g. family, hobby); *members*: the list of members of the network; *events*: the events associated with the network, with a further specification into *type* (providing information on the type of event), *location*, and *time*.

The initial context model is domain-independent, and thus has to be refined according to the application domain considered. At run-time the model is supported by a Context Manager, a middleware component that homogeneously integrates context information coming from the different contextual sources (sensors, devices, etc.).

3.3 Action classification

We have also carried out a classification of the possible actions that can be associated with the personalization rules. As for the context model, such actions need to be refined when a specific domain and application is addressed.
In particular, we have identified a set of action types: those performed in *appliances* (to change the state of some actuator); *user interface modifications (*to change the presentation, content or navigation of the personalized application user interface); *user interface distribution* (how the application user interface should be distributed across multiple devices); *functionalities* (some external service that is accessed); *alarms* (to highlight some potentially dangerous situations); and *reminders* (to indicate some task that should be accomplished).
In the customization phase such actions need to be tailored in order to address the specific services, devices, and appliances that are available in the target context of use.

3.4 Meta-design of IoT context-dependent applications

The main paradigm of EUD is based on the idea of tailorability, namely how end users can be provided with means to adapt software to their own goals in different contexts of use (e.g. at work, at home). So, EUD aims to soften the boundaries between end users and professional developers as well as between “in use” and “development”. Indeed, we need to clearly distinguish between end-user centered design and end-user customization at runtime. While the first one aims to address the users’ specific requirements, the second one encourages more sophisticated opportunities for tailoring software artefacts in the use context. As it has been already highlighted by [Fisher et al., 2004] meta-design is a way to promote the latter idea as it transcends i)professionally-dominated design (which works only for people with the same interests/background/knowledge); ii)user-centered design, which analyses and understands the needs of users at design time; and iii) participatory design, which involves users more deeply in the design process by enabling them to propose design alternatives. All of them tend to focus on system

development at design time by bringing developers and users together to envision the contexts of use whereas meta-design allows design activities to take place at use time.

In our approach the design and development process is structured into three phases (see Figure 1). The first one is carried out by professional developers who are in charge of developing the EUD software, which will allow for customising applications in a context-dependent manner. The EUD software provided by the developers will be general-purpose and domain-independent. As such it will need suitable customisation and tailoring to address the specific issues associated with a specific domain.

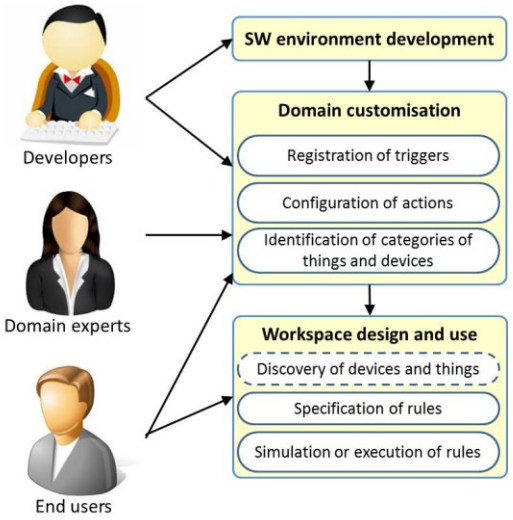


Figure 1: Steps in the proposed meta-design approach

This second phase will be carried out by domain experts (with the help of the developers), who will customise the EUD tool to a specific domain. In order to do this, the following activities should be performed: identification of domain-specific triggers; e.g. for the smart home domain it will include the registration of events and conditions sensed from appliances that are typically available in the home; identification of domain-dependent actions e.g. turn on/off TV, and identification of categories of things and devices relevant in the domain (e.g. TV, tablets, lamps, etc.). After this phase, the outcome is a tool tailored for the specific domain, and, as such, it uses a specific vocabulary typical of that domain. However, a further customisation step is needed when we come at the level of end users, who need a tool that allow them to control real objects in their real contexts of use (e.g. the home). In this customisation step the specific instances of real things, devices, users that exist in the user's real context are automatically discovered and identified by the Context Manager, so that they can be referred by the tool. Thus, we provide an original solution for domain-specific customization in meta-design approaches, since previous work [Sutcliffe and Papamargaritis 2014; Desolda et al., 2015] did not consider the context of use in this type of approach.
After this, end users can specify their rules by referring to triggers and actions actually available in their specific context of use. Thus, in the case that their home is addressed, rules will actually refer to the concrete smart things, appliances, and device functionalities existing in their home. Finally, by using the tool, users can

check the rules (e.g. by simulating them) in order to identify possible errors (e.g. some missing parts) or conflicts in their specifications, or directly execute them.

4. ARCHITECTURE

In the meta-design approach proposed, the purpose is to create EUD authoring environments customized for the various application domains considered. An overview of the general solution architecture is depicted in Figure 2, which highlights the main modules and communications among them.

First, a domain specific context model is refined with the support of domain experts (1) since it requires the knowledge of the application domain characteristics (which contextual aspects to consider and their level of detail). The context model is saved in the Context Manager, which builds its data structures accordingly. The result of this phase is a description of the contextual classes and attributes that can be exploited to define rule triggers in the authoring tool. Then the Authoring Tool needs to be customized as well: it loads the domain specific context model (2a) in order to present the structure of contextual elements to consider when editing the triggers. Likewise, the possible actions are configured starting from a generic classification, and then are customized for the specific application considered (2b). At this point the authoring environment is configured in such a way to allow end users to edit and save the rules by defining triggers in terms of events/conditions related to the relevant contextual attributes and actions.

The saved rules are specified in JSON and sent to the Adaptation Engine (3). Then, when the application is activated it subscribes to the Adaptation Engine (4a), which in turn subscribes to the Context Manager for the events/conditions associated with the rules for that application (4b). Sensors or external services continuously update the Context Server by providing actual data (5). For each previously subscribed event and condition that occurs in the current context, the Context Manager notifies the Adaptation Engine (6), which extracts the list of actions from the verified rules (i.e. the rules having the 'trigger' part verified) and pushes them to the application (7) and then to the relevant Appliances, when necessary (8).

Further details on the single modules are given in the following.

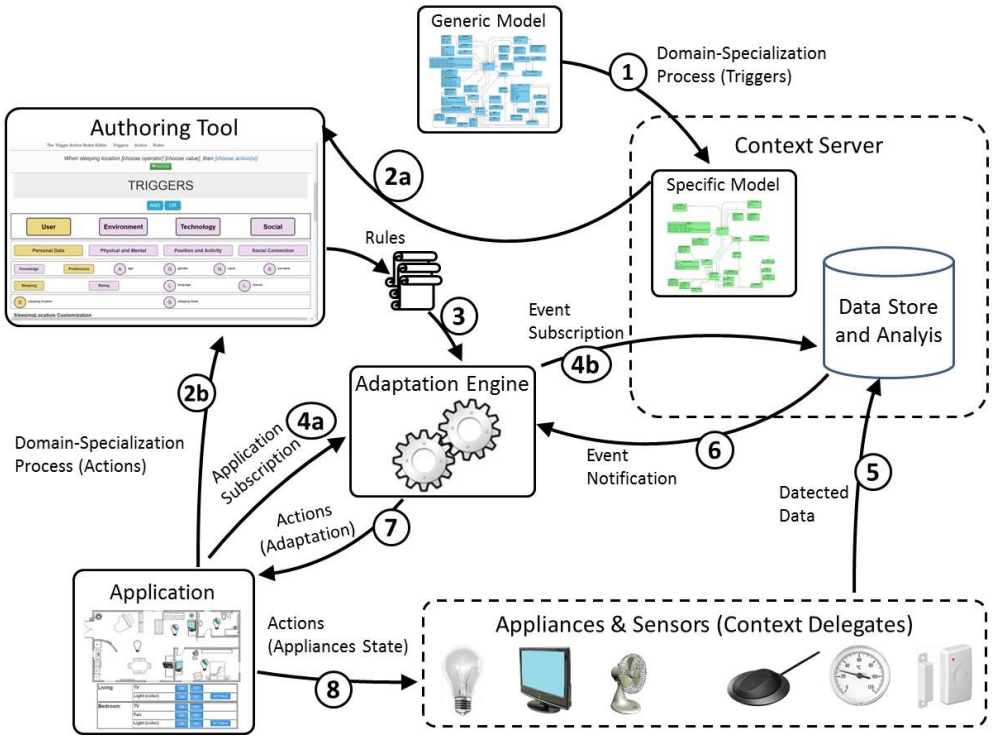


Figure 2: The Environment Architecture

The Context Manager provides all the functionalities to collect data from external contextual sources, store and analyse them. It is composed of a Context Server and a set of Context Delegates (see Figure 2). At run time contextual information is organized in a set of instances of interconnected classes, where each class can have one or more attributes.

Data arrive from various sources, such as sensors (temperature, noise, light, doors/windows closure, power absorption, etc.) or external services (e.g. weather forecast). Connection to sensors and data gathering is made by context delegates specifically developed for the type of sensor to interface with and for the type of platform where the software is running. For instance, a context delegate connecting to a power absorption sensor can be deployed on a desktop PC, while a smartphone can host another software module that detects the environment noise by interfacing to the device's microphone. Such software modules, although potentially distributed on various devices, are considered to be part of the overall context management functionalities, and implicitly manage the appearance/disappearance of resources/sensors in the environment. This is possible because the Context Delegates are expected to update sensors values at specific time intervals, and the entities stored in the Context Server are characterized by an update timeout. Thus, if a sensor fails, a timeout exception is raised on the entity referred by the sensor. For instance, if the temperature sensor of the living room is turned off or is uninstalled, then the Context Delegate responsible for that sensor stops updating the temperature entity for the living room on the Context Server. The subsequent update timeout will imply the entity deletion on the Context Server. However, as soon as the temperature sensor resumes (e.g. it is plugged in again or the battery is replaced,

etc.), upon receiving the first update, the Context Server will recreate the associated entity and update its attributes.

Generic Context Model. A context model, saved as XML data schema, defines the types of data the Context Manager can maintain and the relationships among such data. The context model is structured in terms of classes and associated attributes and provides a general reference vocabulary.

Domain-specific Context Model. A specific, domain-oriented version of the context model is created to better suit the needs of a certain application domain. The domain-oriented context model, rather than replacing the generic one is a refinement of it, and it acts as an intermediary between the end-user Authoring Tool and the Context Manager. The domain-oriented context model is used as input to structure the domain-dependent Authoring Tool, in such a way to parameterize the user interface for rule triggers selection accordingly.
The creation of a domain-oriented context model from the generic context model is performed with the support of domain experts. There is no limit to the definition of new classes in the domain-dependent context model, neither in terms of number nor in terms of names or connections among them. For each class, an arbitrary number of attributes can be defined as well, without any naming restrictions. This distinction between generic and domain-specific context model, facilitates the reusability of the context-dependent platform. For example, the *User* class can be modelled differently in different applications, each one with an associated domain-specific context model, which refers to the same generic context model. For instance, in the smart retail domain, the generic *User* is modelled as *Customer*. Most of the original data structures with their detail level are maintained. *Personal Data*, including *Nutritional Preferences*, *Age*, *Education Level*, *Work*, are kept since they are relevant for the domain. Such contextual aspects are useful for taking into account the actual needs of the customer and provide them with tailored commercial recommendations. In the Ambient assisted living domain, the generic *User* is modelled as *Patient*. Aspects such as *Disabilities* (e.g. *motor impairment*, *blindness*) and *Diseases* (e.g. *diabetes*, *hypertension*) are included in the specific context model. All the user's physiological parameters (e.g. hearth rate, blood pressure) are also included for monitoring purposes. Other ones, such as *Education Level* or *Work*, are not particularly relevant for the domain, thus they are not included in the specific model.

Besides choosing which classes and attributes to consider and how to define them (including the possibility to rename some of them with more relevant terms for the domain considered), the domain-specific model also offers the possibility of introducing new connections between classes. By adding further transversal connections, the structure of the context model, which is mainly tree-like, becomes more similar to a graph. For example, the location entity can refer both to users and to devices or objects.
The deployed instances of the context managers refer to the relevant domain specific context models.

Contextual Information Store and Analysis. Contextual data coming from multiple sources are kept on a repository with two main purposes. One is the synchronous access on request, e.g. an application that accesses the user profile in order to initialize the UI based on user's preferences when starting. The other is to

asynchronously notify modules that have previously subscribed to relevant events. Asynchronous notification is enabled by the continuous analysis that the internal routines of the Context Manager perform on the Contextual Data Store content. More in detail, for each variation of any parameter involved by an event subscription, the Context Manager checks whether the constraints (which can be associated with the event) are met. In the positive case, a notification is sent to the subscriber. In the current architecture, the subscriber module is the Adaptation Engine. It extracts events and conditions from the triggers defined in the rules created through the authoring environment, and accordingly subscribes to the Context Manager.

Sensors. In our architecture, any source that provides contextual data to the Context Manager is in general referred to as a sensor. Digital thermometers, light/noise sensors, windows/doors closure sensors, and other actual (physical) sensors do interface with the core of the Context Manager via dedicated software modules (context delegates) which are aware of the hardware and low-level protocol peculiarities. Such context delegates periodically query the associated sensors and save the response value on the Context Manager.
External services providing weather forecast, user's activity on the social network and other digital data, can also be considered as sensors because they provide input to the Contextual Data Store. An example of adapter module for a virtual sensor could be a routine that analyses the profile of the user's social network and estimate the level of activity. Another one is the routine that queries the weather forecast web service every day in the morning. These parameters can be involved in the rules and trigger some action in the smart environment: changing the illumination colour according to the user's mood inferred from their social activity, starting the garden irrigation if no rain is expected in the next couple of days.

Adaptation Engine. The module in charge of applying adaptation actions is the Adaptation Engine. At configuration time it receives the set of active rules from the Authoring Tool along with the addresses of the relevant Context Server and the concerned Web application. At run time it loads the rule triggers and subscribes to the Context Manager for the specified events and conditions. When a notification is received (e.g. the user enters home), the Adaptation Engine extracts the list of actions from the related rule and send them to the concerned application for being properly applied. The actions determine updates in the home application (e.g., reorganizing the layout, activating functionalities, etc.) and/or changes in the state of the appliances (e.g., switching on the radio, varying light intensity, etc.).

Appliances. Any device or object able to be remotely controlled is an appliance. Examples are digital TVs, intelligent lights (e.g.: Philips HUE bulbs), actuators for windows, dimmers and electronic relays.

5. THE TRIGGER-ACTION RULE EDITOR

The Authoring Tool is Web-based and enables the creation of (multiple) trigger-action rules in an intuitive manner. It has been designed by taking into account the requirements previously indicated. Thus, it is flexible in the order in which rules can be created: users can start either from triggers or from actions (see Figure 3), and it provides the possibility to reuse previously specified rules as a starting point in order to create new ones.

Figure 3: The User Interface presented to users at the beginning of the session

Each rule is composed of two parts: a trigger part, and an action part. Thus, its basic structure is the following one:

IF/WHEN <*trigger_expression*> DO <*action_expression*>.

The *trigger_expression* defines the event(s) and/or the condition(s) that activate the rule application. The *action_expression* defines the action(s) that should be carried out when the rule is triggered.

Trigger_expression and *action_expression* can be either an elementary or a composite expression. In the previous sections we have introduced their general conceptualisation, i.e. the various aspects that a trigger and an action can describe. We also provide the possibility of combining multiple triggers and/or multiple actions. As both the evaluation of events and the evaluation of conditions result in Boolean values, multiple triggers can be combined by using basic Boolean operators, namely AND and OR. It is also possible to use the NOT operator in the definition of conditions. It is also possible to define sets of actions to be sequentially performed.

In the editor, the selection of the relevant concepts is performed by navigating in the hierarchy of concepts associated with each contextual dimension (user, environment, technology, social). For this purpose, each contextual dimension is refined by means of a number of conceptual levels until basic elements are reached. In order to show only the relevant elements to the user, the refinements are presented in an interactive manner (i.e. only the decomposition of the element currently selected by the user is expanded), and highlighting the element(s) selected by the user through a different colour. The selected concepts are highlighted in the tool by the yellow colour. Figure 4 shows an example in which the yellow rectangles represent the path from the dimension to the event/condition aspect: User -> Position and Activity -> Position -> Relative Position. This was a design choice aimed at reducing the cognitive load for the user (limit the number of elements to be visualised at a certain point, so that users can focus only on the actually relevant ones).

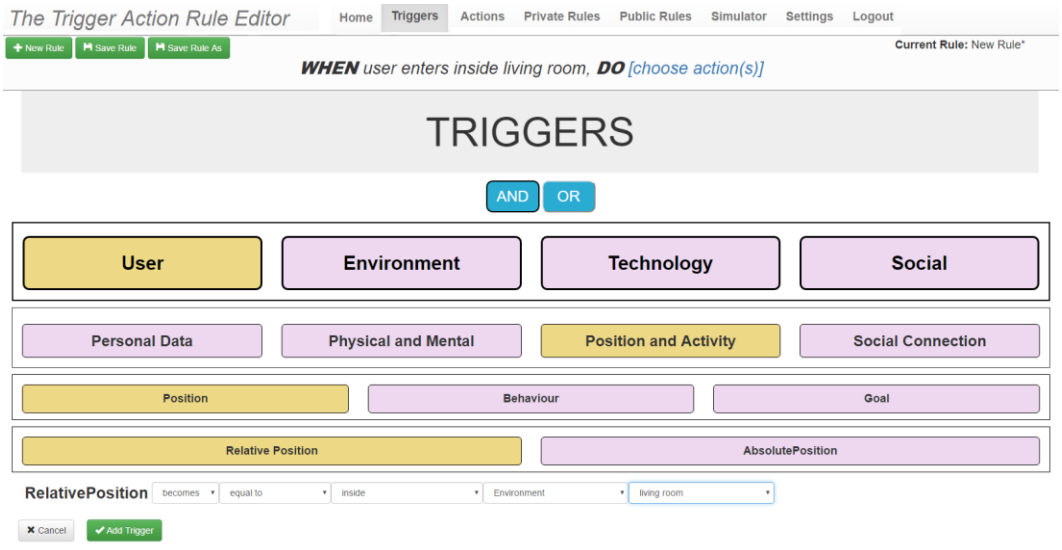


Figure 4: Editing a trigger expression in the tool

Basic elements are represented differently from the elements that are in the higher levels of the decomposition: indeed, the latter ones are presented by using a rectangular icon with the name of the concerned category inside, while the first ones are shown by using circled shapes with 1-2 for letters identifying them. This has been done in order to save space when the tool is accessed from a mobile device and also to clearly distinguish between the leaves and other higher-level elements in the hierarchy. The user interface is responsive and it has been implemented using the Bootstrap framework. When users choose a basic element, the tool provides them with the support for specifying the attributes, the operators and corresponding values according to which the rule will be built. A video showing interactive rule editing is available at https://www.youtube.com/watch?v=KnYX0CTMwlI

On the top there is a continuous feedback in a subset of natural language indicating the current edited rule in an easy-to-understand language. The distinction between events and conditions is highlighted by using different keywords: “WHEN” is used for events, whereas “IF” is used for conditions.

Indeed, the tool provides users with means to distinguish between events and conditions: users specify events by using the “becomes” operator (which models the fact that users are interested in when a certain attribute changes its value), otherwise they will use the “is” operator. If users specify a condition in the trigger part, “IF” keyword is used in the corresponding natural language phrase, alternatively (i.e. in case of events) “WHEN” is used. These two keywords are automatically added in the natural language expression, according to what the user interactively specifies in the rule. Figure 4 shows an example of use of the “becomes” operator. The natural language description of the rules is interactive in its composing parts, i.e. it is possible to select the various parts and edit them. Thus, the tool reports “when user enters the kitchen” if the users have indicated that they are interested in the event (e.g. the moment when the user position changes to “kitchen”), while it provides “if user is in the kitchen” when users indicate interest in the condition of being in the kitchen. Another important point is the fact that, if there are multiple objects of the same type (e.g. typically there are multiple lights in

a house, at least one for each room), the tool allows users to select a single action to act on all the objects of the same type at the same time, so preventing the user from tediously specifying each single action separately.

When different Boolean operators are included in a rule, the editor automatically adds brackets in proper places and then it highlights in the resulting natural language expression how it will be interpreted by the system, so as to reduce the possibility of its ambiguous interpretation. The brackets are automatically added by following the logical operator precedence commonly used in programming languages. Nonetheless, if users are not satisfied by the proposed order, they can change it in an interactive manner, by using the features of the tool.

In a similar way users can specify the desired actions. The authoring environment supports the classification of possible actions introduced in section 3.3. When one of the main action types is selected, the tool shows the supported corresponding application-dependent actions (Figure 5 shows an example in which the action is "set living room light colour to white").

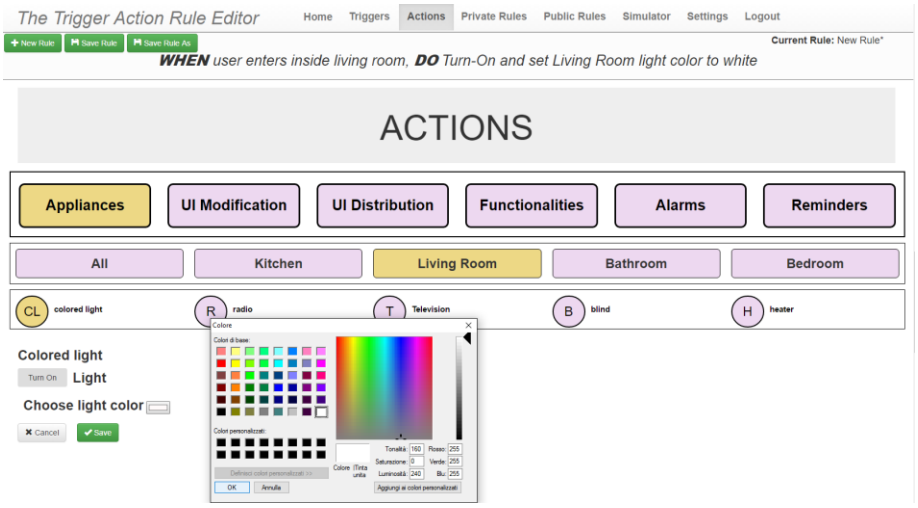


Figure 5: Interactively specifying an action expression by using the tool

When some rules are created, they can be saved for possible reuse in different scenarios or by other users (Figure 6). When the user wants to re-use a previously created rule s/he first has to select it, and then activate its editing: after this the tool allows the user to change the rule by selecting its constituent parts in the natural language section and updating some values or attributes in its specification. Furthermore, it is possible to add some more triggers/actions, or even delete some parts (e.g. one of the actions and/or one of the triggers).

In addition, the user can select, from the set of rules already saved, the ones she wants to actually execute in the current scenario. In the settings it is indicated the actual instance of the Context Manager with which to communicate. After selecting the ones to actually apply, the user will be notified about possible conflicts in the actions associated with the rules currently selected. For example, the rules "In the morning lights should be off" and "When I get home lights should be on" can be in

potential conflict depending on the actual time the user gets home. In such cases, priorities are used to solve the conflict. The conflicts are detected by analysing the objects manipulated by the actions (they can be appliances or parts of the user interface application), as well as the involved triggers.

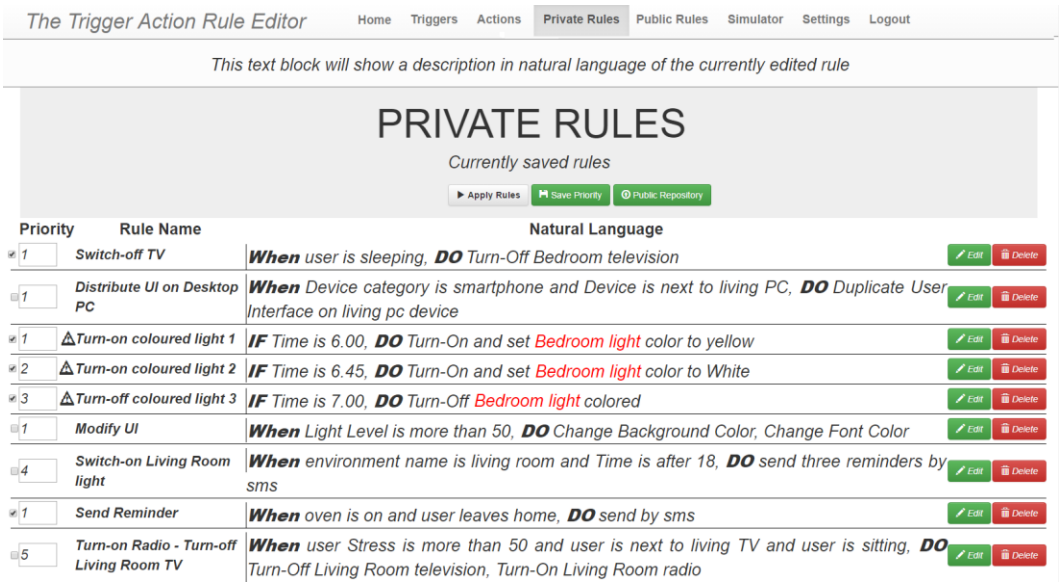


Figure 6: Example of a set of saved rules.

If a rule refers to elements that are no longer available, such as a light that for some reason no longer works, the corresponding part in the rule is highlighted in red to indicate the problem that does not allow the rule to be executed. It is also possible to add the rules in a public repository so that other users can download and reuse them.

Another feature of our editor allows for simulating the execution of a rule in a simulated context. This simulated context is built by the user by means of interactively specifying a number of contextual aspects the user is currently interested, in terms of either conditions or events (e.g. user position becomes kitchen, user is Alan, date is tomorrow afternoon, living room TV state becomes off, etc.). Then, the tool checks whether the selected rules could be triggered in the simulated context (i.e. rule conditions and events are verified) and it colours the verified rules in green and the others in pink (see example in Figure 7). In the rules not verified it also highlights the events that have not occurred or the conditions that are not verified in the simulation, thus providing users with support for understanding why they are not triggered.

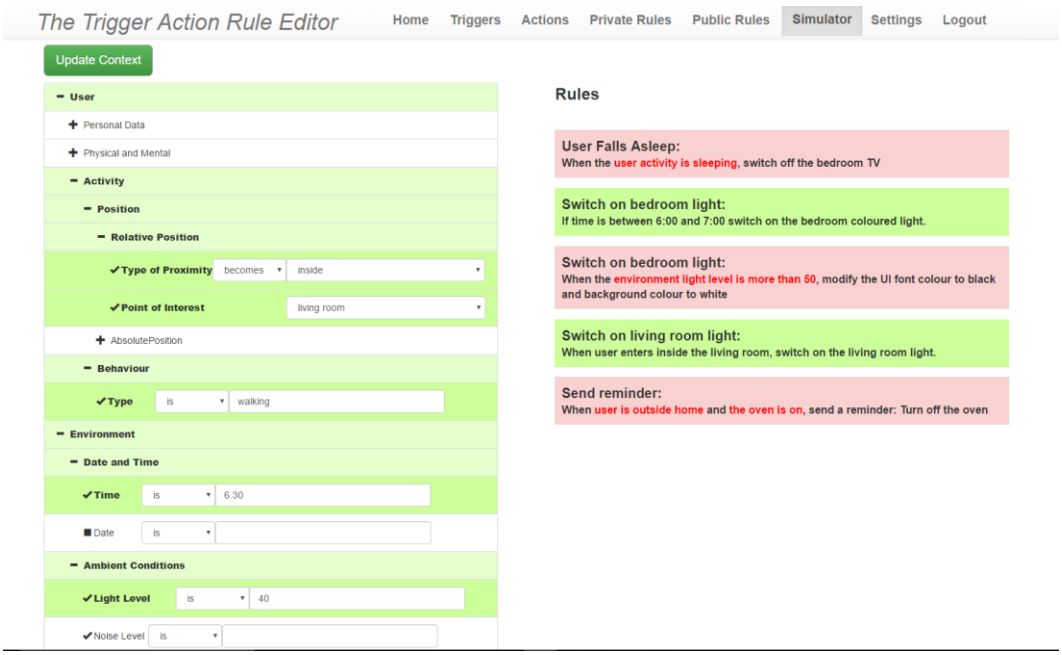


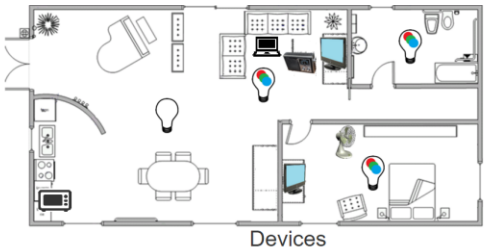
Figure 7: Interactive simulation for identifying the rules that can be triggered.

In addition, it is also possible to have rules directly executed in the current context of use, to verify whether, in the current real conditions, they would trigger the right effect. However, for the time being, our tool is not able to support sophisticated reasoning about the saved rules in such a way to better handle specific cases (e.g. when two rules are dependent each other because e.g. the action of one rule is the trigger of the other rule).

When comparing our approach with IFTTT, the latter seems to exploit a rule-based metaphor of a similar level of intuitiveness. However, one of the main disadvantages of IFTTT is its limited expressivity since it only allows selection of a single trigger and action per rule from predefined lists. In addition, IFTTT requires a sequential definition of trigger and actions (first event and then action editing) when building the rules, whereas in our approach any order is possible. Moreover, IFTTT provides support to access various applications and services, however users cannot extend them to create rules for others, while we provide the possibility to configure the authoring tool for new Web applications. Finally, while our solution provides a logical classification of triggers and actions which can be interactively explored by the user, IFTTT lists all the services in alphabetical order, which makes it difficult to find the right trigger/action. IFTTT also allows doing a search among the possible services available, but this is not very effective when the names of the involved services are not precisely known.

6. AN EXAMPLE APPLICATION

In the smart home domain, users typically customise and control the appliances through a home controller application. We have considered a home application (Figure 8) with the main aim of displaying sensor values and directly controlling appliances. The initial application has limited possibilities in terms of personalization, and we want to show how the approach proposed allows end users to dynamically change its possibilities by editing relevant rules.



Sensors

Temperature (C°)	19°
Light (lx)	52 lx
Noise (dB)	50 dB
Entrance door (open/closed)	Closed
Balcony door (open/closed)	Closed
User Stress Level	30
Is user Sleeping?	false
Presence in living room	true

Devices

Living Room	TV	ON	OFF	
	Light (color)	ON	OFF	SETTINGS
	Radio	ON	OFF	
	PC (Desktop)	ON	OFF	
Kitchen	Light	ON	OFF	
	Oven	ON	OFF	
Bedroom	TV	ON	OFF	
	Fan	ON	OFF	
	Light (color)	ON	OFF	SETTINGS
Bathroom	Light (color)	ON	OFF	SETTINGS

Figure 8: The main menu of the example home controller application.

The first step was to create the domain-dependent context model. We kept the four general contextual dimensions (User, Environment, Technology, and Social). We identified, for each general class, the list of relevant attributes. We re-organized the set of attributes in a new class structure (i.e. by removing existing classes and adding new ones). We finally renamed some attributes and classes for intuitiveness purposes.

The *User* dimension mainly remained unchanged. The *Environment* dimension has been specialized for the home domain in such a way to consider the typical home environments (e.g. kitchen, living room, bedroom). *Technology* includes common home appliances such as TV, fridge, radio. The *Social* dimension structures various possible social relationships, some general such as friendship, relatives, belonging to special groups of interest, and some more specific to the home domain (such as neighboured). We also configured the types of actions that can be relevant in terms of user interface modifications. They mainly refer to changes in user interface attributes or distributions of parts of the user interface to other devices.

Then, we obtained the EUD authoring environment for the considered domain and application, which can be used to create interactively rules such as “When I enter the living room and it is evening then the lights and the TV should be turned on”.

7. USER TEST

We performed a user test to assess the usability of our approach for specifying trigger-action rules with the authoring environment customized for the home domain

and the application considered, in order to evaluate whether a non-programmer user could easily create rules in the considered domain and exploiting the proposed tool

7.1 Participants

18 participants (4 male) with age ranging between 20 and 60 (mean 38,7, median 36, std. dev. 11,1) were involved in the user study.
Participants were recruited through messages on the mailing list of the research centre of the authors' lab. They were workers in administrative roles, or relatives/friends of workers. None of them was involved in our research and development work. Six users held a High School degree, 5 a Master, 4 a PhD and 3 a Bachelor. Twelve users had no knowledge of programming languages and 6 had only low knowledge.
The great majority of them had never used any customization tool for smart environments. One user was familiar with IFTTT, another one had some experience with IFTTT and Tasker. The vast majority of users were not familiar at all with smart environments or appliances such as Philips Hue lights.
As compensation for their time and effort, participants received a small backpack with the logo of our laboratory (estimated value around 10 USD).

7.2 Test organization

The participants received an introduction to the study consisting in reading motivations and aims. They were also provided with some high level description on the authoring tool and the trigger-action rule structure.
They watched a short video showing the authoring tool capabilities and some example interactions for building trigger-action rules. Then, they could interact with the authoring tool for few minutes, in order to familiarize with the rule definition mechanisms.
Before starting the interaction, they were asked to freely provide a short list of trigger-action rules they would define for their home environment, without any constraint neither on the triggers nor on the actions. After that, they were provided with the description of the scenario and the list of behaviours (i.e. rules) to create through the authoring tool. Each rule was described in natural language in terms of trigger(s) and action(s).
The rules were identified in such a way to cover most of the aspects addressed by triggers (namely: user, environment, technology, social) and actions (namely: appliances, UI-related aspects, functionalities, alarms, reminders). In addition, we also varied the level of complexity of the behaviour to specify so that people had to specify rules having simple triggers (see e.g. 'a' behaviour in Table 2) and rules including complex triggers (i.e. those resulting from the combination of multiple triggers, see e.g. 'g' behaviour in Table 2); similarly, we considered rules including single actions (see e.g. 'a', 'e', and 'f' behaviour) vs. other rules more than one actions to execute (see e.g. 'd', and 'g' behaviour).

If we want to consider how IFTTT would have managed the rules considered in the test, it is possible to say that, since IFTTT is not able to handle logical Boolean operators appearing in complex triggers (e.g. AND), all the rules in which a complex trigger appears with an AND operator are not supported by it (see e.g. f) and g) rules in our test). Instead, the rules in which multiple actions appear could have been handled by IFTTT only by setting up multiple recipes (see for example c) and d) rules).

The scenario description which was given to users is the following one.
Your house is equipped with a set of sensors and appliances. The sensors detect contextual aspects such as environmental parameters, presence/motion, state of doors/windows, etc. Appliances are actuators and devices that can be remotely controlled. With the proposed authoring tool you can create trigger-action rules (see the example video) to control devices based on events that occur and conditions on contextual aspects.
You are requested to implement the behaviours specified in the following. Note that more than one rule may be needed to implement each behaviour. Each rule must be saved before starting editing the next one (the requested behaviours/rules are listed in Table 2).

Table 2: The rules considered in the user test

a	When the user falls asleep, switch off the bedroom TV.
b	When a smartphone device is close to the Desktop PC of the living room, duplicate the UI of the smartphone in the Desktop PC.
c	The coloured light in the bedroom must be on between 6:00 and 7:00. It must emit yellow light between 6:00 and 6:45, and then white light till 7:00.
d	When the environment light level exceeds value 50, modify the UI by setting black font colour and white background colour.
e	As soon as presence is detected in the living room, switch on the living room light.
f	When the user leaves home and the oven is on, send a reminder to turn it off.
g	When the user stress level exceeds value 50 and the user is sitting close to the living room TV, then turn off the living room TV and turn on the living room radio.

After the interaction session, users filled in an online questionnaire indicating their expertise in programming, whether they had previously used any system for building trigger-action rules, and some personal data (age, gender, education). They also rated, on a 1-7 Likert scale, some aspects of the proposed environment and provided observations on positive/negative aspects they noticed on the assessed system and recommendations for its possible improvements.

7.3 Results

We wanted to estimate to what extent the complexity of the rule(s) supporting a specific behaviour would correlate with the effort needed to define the rule and their resulting quality. The execution order was thus randomised for each participant, in order to minimise the learning effect.

The completion time was recorded for each participant and for each rule. In the following, the term "rule" is generically used to indicate one of the a-g rules, since *c* was the only behaviour requiring the definition of more than one rule. A moderator observed the participants interacting with the authoring tool, annotating any issue, remark, question that they had. The screencast of the interaction was also recorded, in order to facilitate the post interaction session analysis.

We analysed the list of example rules by the participants (i.e. the rules they would apply to their smart home, if they had one), and found out that 12 participants found it important to be able to define rules with a complex trigger or action.
We collected a total of 55 rules. In the following, some example rules with various degree of complexity given by the participants are reported:

Simple rule (single trigger and action)
- At midnight close the blinds (P1).
- When the environment light level drops below a threshold, turn on the light (P3).
- When leaving home, send a reminder to close the gas valve (P14).

Complex trigger and simple action
- At night, switch on the corridor light, but only when presence is detected (P9).
- When getting home in working days, the blinds must keep closed in winter and open in summer (P15).
- When I am out of home and it rains, close the blinds (P16).

Simple trigger and complex action
- Based on the humidity, ventilate the house and control the dehumidifier (to prevent the formation of mould) (P2).
- Control curtains, blinds and external lights according to the sunlight (P5).

Complex trigger and complex action
- When I am out of home and on holiday, switch on at regular intervals some of the inner lights in order to simulate presence at home (P13).
- When I leave home, if the pet is at home, a blind should be kept partially open and a door open (P18).

All such rules could have been specified with the proposed tool.

The aspects in the following were rated on a 1-7 Likert scale (med is the median value). The lowest and highest scores were associated to judgement labels depending on the question (e.g.: 1 = very bad, 7 = very good; 1 = low exhaustiveness, 7 = high exhaustiveness; 1 = low usefulness, 7 = high usefulness).
- Usability of the trigger selection mechanism (min: 3, max: 6, med: 5)
- Usability of the action selection mechanism (min: 3, max: 6, med: 6)
- Usability, in general, of the rule-based approach (min: 3, max: 7, med: 6)
- Exhaustiveness of the set of events that can be specified (min: 4, max: 7, med: 5)
- Exhaustiveness of the set of actions that can be specified (min: 4, max: 7, med: 6)
- Usability of the mechanism for reusing previously saved rules (min: 4, max: 7, med: 6)
- Usefulness of the rule description in natural language (min: 3, max: 7, med: 6)
- Usefulness of the approach to make a domotic home context-dependent (min: 4, max: 7, med: 5,5[6])

[6] The number of values was even with the same number of 5s and 6s.

The stacked bar chart in Figure 9 shows, for each aspect, the percentage of participants that have provided each score. None of the aspects was rated with 1 or 2.

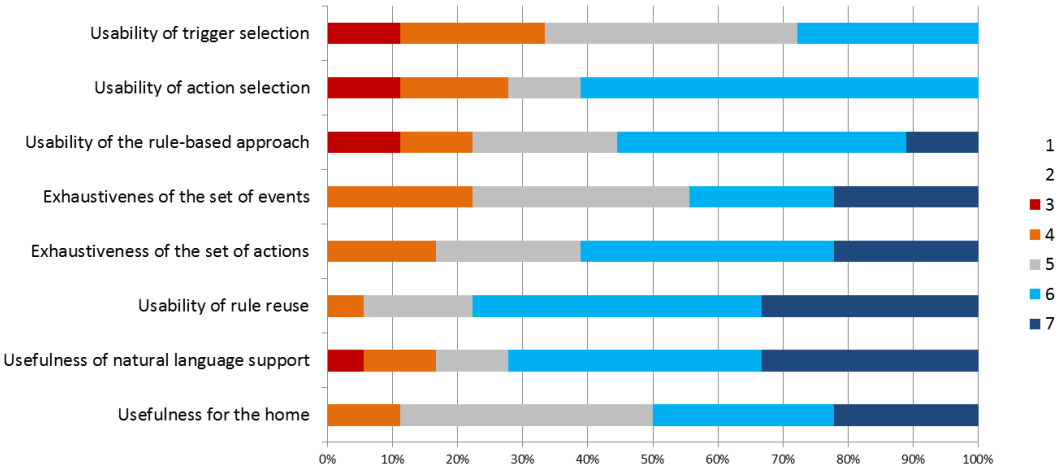


Figure 9: Stacked bar chart for participants' scores percentage.

The participants also answered the following open-ended questions.

Do you have any suggestion to improve the usability of this approach?
Five users recommended simplifying the trigger selection part. The motivation was to make it easier the search of the desired attribute. Three users suggested to make more intuitive the action selection mechanism too.

Do you have any suggestion about elements to add/remove to/from the set of events?
One user would have liked the possibility of customising the set of events by creating new ones. Another user suggested a finer control over the indoor localization (e.g. by specifying iBeacons). One user found that the set of events is already too wide.

Do you have any suggestions about elements to add/remove to/from the set of actions?
We received two opposite comments by two participants with opposite views: one recommended simplifying the set of actions, the other one indicated to customise the set of actions by adding new ones to the interface.

Do you have any comments on the way the rules are described in natural language?
One user stated the support is really helpful. Another one complained that the language was not so natural and gave the lowest score to the usefulness of the natural language support: she believed the language was not natural at all, being still too structured. A third one recommended improving the language naturalness by making sentences shorter by avoiding repetitions, e.g. "if object is oven and object state is on" should turn into "if oven is on".

State three positive aspects of the proposed Authoring Tool.
All the users expressed some positive aspects. Examples of positive aspects cited are: usefulness; possibility of creating complex rules; relative simplicity of use and easiness in getting familiar while interacting with it for the first time (i.e. easy to learn); exhaustiveness of the range of rules that can be created; quick; clear graphical

elements; clear distinction between trigger and action areas. Six users liked the natural language description for the edited rule.

State three negative aspects of the proposed Authoring Tool.
All but one user mentioned negative aspects, such as: issues in distinguishing the difference between "is" and "becomes"; too many selectable elements; some ambiguities across the terms used (e.g., technology/device/object); issues in identifying where to find what is needed; not intuitive in some circumstances; requires some practice to be used quickly; missing constructs to create alternative rules (else, elseIf, switch).

Do you have any general suggestions to improve the Authoring Tool?
The most frequent suggestions were due to the wish of simplifying the rule composition: adding a sort of wizard mode for supporting beginners or presenting a set of predefined rules to be modified as needed; a more graphical approach for selecting triggers and actions through associated icons and, possibly, a clickable map of the house; add a "search engine" to quickly find a trigger; further separate triggers and actions, by e.g. using different styles.

7.4 Errors

An error in defining the rule is due to a difference between the result of the execution of the rule defined by the participant, and the result expected by the correct rule.
We have classified such errors into three main categories: minor, moderate, severe.
Minor errors were due to mistakes in the values of conditions or actions, such as a wrong values associated to a trigger or to an action parameter. Examples of minor errors made by the participants are: specifying "personal" instead of "smartphone" as type of device (rule b); specifying "besides" instead of "next" (rule g).
Moderate errors were caused by wrong choices in the type of trigger or action but the resulting rules could still work, but performing only partially the requested behaviour (either because some case would not be considered or some unrequested action would be carried out). Examples were: setting an "attention" level instead of specifying "activity = sleeping" for the sleeping condition of the user (rule a); setting "light level = 50" instead of "light level > 50" (rule d).
Severe errors characterised rules that would not provide the requested behaviour at all due to serious issues in the specification of the trigger and/or the action. Examples were: duplicating the UI from the smartphone to the smartphone (rule b); setting "description=on" instead of "state=on" (rule f).
Table 2 reports the three types of errors for each rule. At rule f, 7 out of 18 participants made an error (3 minor, 2 moderate and 2 severe). The letters into brackets indicate for each rule whether it is characterised by complex (C) or simple (S) trigger and action, respectively.
However, there not seem to be a relationship between the complexity of the rule and the number of users that made errors, because only one participant made a moderate error in rule g (with complex trigger and complex action), while we had two errors in rule a (simple trigger and simple action). In general, there are various possible types of trigger combinations. In the rules that we used in the test the users found no particular problems. We can see that specific compositions for particular cases may not be completely intuitive for some people.

In total, 126 rules (7 x 18 participants) were defined, and 15 of them had errors. Thus 111 of the defined rules would work perfectly, 3 would provide a slightly different behaviour from the desired one, 7 would work only in some cases and 5 would be useless to model the requested behaviour. Since the purpose of the test was to assess the usability of the rule editor tool we decided not to use the simulator tool in the test.

Table 3: Errors in User Test

	a (S,S)	b (C, S)	c (C, S)	d (S, S)	e (C, S)	f (C, S)	g (C, C)
Minor	1	0	1	0	0	1	0
Moderate	1	0	0	0	3	2	1
Severe	0	2	1	0	0	2	0

7.5 Task completion time

Figure 9 reports minimum, maximum, mean and standard deviation for the time required to define each rule.

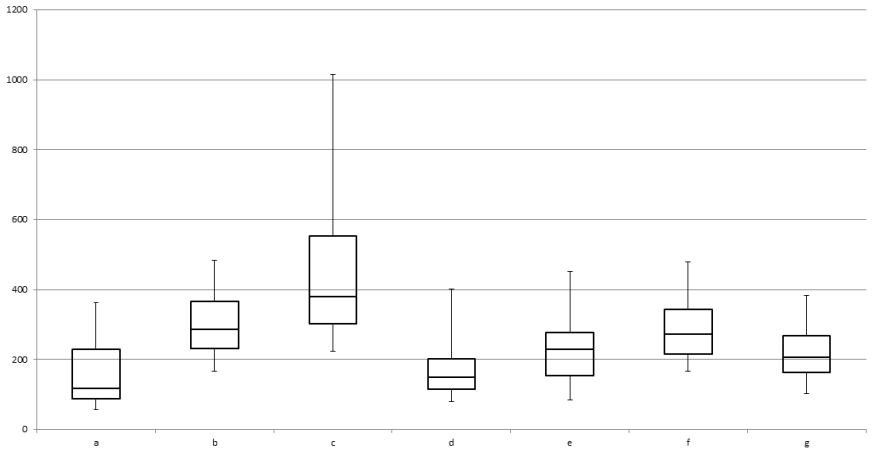


Figure 10: Task time for a-g rules. Vertical axis indicates the completion time.

Simple rules (a, d) were on average quicker to define. The rule that required longer was c, which was difficult for two reasons: 1) it was expressed in a different way with respect to the others; 2) it required two or more rules to be implemented. The way it was expressed was misleading as its description started with an action rather than with the trigger. Many participants thus started the composition of the rule by looking for the "coloured light of the bedroom" rather than for the time aspect. This confusion made participants waste time. In addition, several participants tried for a while to model the behaviour in a single rule. Most of them, after seeing the wrong rule description, understood the mistake by themselves. Some others gave up and asked the moderator whether it was possible to put all the information on a single rule. Upon receiving a negative answer, all but one participant were able to define the rules. It is interesting that, while we thought point c as consisting of two rules with complex trigger, some users implemented it through three rules with single

trigger: instead of keeping light yellow between 06:00 and 06:45 and white between 06:45 and 07:00, they turned on the yellow light at 6:00, then switched it into white at 06:45 and finally turned it off at 07:00.
We considered the task completion times to see whether the participants, over the test, tended to become faster in specifying the rules. We sorted the task completion times according to the actual execution order, which was randomised. We computed, for each user, the mean task completion time for the first four tasks and for the last three tasks. Eleven users (61,1%) were faster, on average, on the last three tasks than in the first four. We then calculated the mean completion time for the first five and the last two tasks. Fourteen users (77,8%) were on average faster in the last two than in the first five. By comparing the mean completion time for the first six tasks with the last task, we saw that 12 users (66,7%) were on average quicker to solve the last task than in the previous ones. This indicates that, independently of the complexity of the rule, users tend to become quicker as they get familiar with the mechanisms of the tool.
The last task, i.e. modification of previously saved rule e, took on average 142 seconds (min: 39, max: 515, std. dev.: 111), which is most probably due to the task simplicity. In that task users had only to modify the action part.

As previously mentioned, we decided to model such a conceptual difference by letting users specify the IS/BECOMES operator option. All the rules that the users had to define were expected to contain at least an event. We observed that participants had some difficulties to choose the proper option in all the rules.

In general, the results are encouraging. Users understood the capabilities of the system and acknowledged the potentialities of the proposed approach. The scores on usefulness, exhaustiveness and usability, on a 1-7 Likert scale, were more than acceptable. This is especially true if we consider that participants had to create various complex rules and none of them had programming skills. The trigger selection part received the least positive scores. When users were finding the needed trigger attributes, we observed that most users started looking for the desired attribute by following a path that made sense for them, i.e. they chose a dimension (e.g. User), picked an intermediate node (e.g. Position and Activity) and continued exploring the trigger structure while thinking aloud. The users that were unable to quickly find the desired attribute at the first glance can be divided into two categories: those who iteratively explored all the possible triggers with the aim to find at some point the desired one; and those who repeatedly looked for the desired trigger under the wrong intermediate nodes. These ones excluded a priori some intermediate nodes of the trigger structure since they felt those nodes would not contain the desired trigger. In order to complete the task, the moderator had to assure that they would have found all the requested triggers within the hierarchy. We believe that, for those users that tend to "give up" after a few unsuccessful searches, an automatic support would be useful. An example is the possibility of specifying a keyword (e.g. the name of an attribute) on a search box for getting the path to the corresponding attribute on the trigger structure. A user suggested that a visual map of all the triggers would be a useful support for novices to quickly find the desired attributes. She also suggested to investigate the use of different colours for visualising the nodes of the various contextual aspects.

Although the scores associated with the trigger selection part were not negative (they were on average between 4 and 5) some participants commented that they would further simplify the trigger selection. A possible way is making easier the structure of the set of triggers, and/or involve domain experts as well as end users in the choice of the elements composing the triggers. It should be noted that modifying the triggers would not require a reimplementation of the tool, but only a modification of the domain-specific context model that is loaded every time the authoring tool is run.

7.6 Discussion and Main Findings

The overall observations made by evaluators and the test results show that the proposed approach can be exploited by the non-programmers involved in the test, who proved able to quite easily grasp the main conceptualisations and interaction mechanisms provided in our approach.

The questionnaire answers indicated that most participants would deploy complex trigger-action rules on a smart environment. Thus, they found it useful the possibility of developing rules with conditions on multiple contextual aspects combined to formulate a trigger with multiple actions. This confirms the current need to investigate solutions for this purpose.

The results of the usability test also show that, despite some difficulties, the majority of the participants (none of them had relevant programming experience) were able to implement most of the requested rules. One of the key advantages of the solution, emerging from the open-ended participants' comments, was the ability to see in real time a preview of the edited rule in natural language. The issues experienced were due to the difficulty in finding some of the requested attributes, since the path in the context model hierarchy for accessing some contextual attributes was not always intuitive. For example, by observing the participants of our study we learnt that, for some users, "sleeping" is not an activity but rather a physical / mental state.

Also several users reported to have had difficulties in interpreting the distinction between events and conditions.

The study also showed that even when users had to specify structured rules (e.g. those including complex triggers and/or combination of actions), they tend to become quicker as they get familiar with the mechanisms of the tool, which could indicate that an increased complexity of the rule does not impact much the rule creation process.

As for the limitations of our work, the rule reasoning can be further elaborated to detect dependencies amongst rules and better manage them. In addition, further empirical user studies for longer periods of time can be useful to better validate the presented approach. In such studies we also plan to carry out specific tests regarding the usability and utility of the simulation support available in the environment. In addition, while in this work we have focused on personalization of Web applications, we intend to investigate the effectiveness of our solution when other types of application implementations (such as mobile native apps) are considered.

8. CONCLUSIONS AND FUTURE WORK

We have presented a solution for enabling end-user customization of context-dependent applications. Our approach is based on the combination of generic and domain specific components. A general purpose context model and a flexible authoring tool go both through a domain specialization process devoted to determining relevant contextual aspects for triggers and possible actions. The resulting artefact is a tool that hides most of the generic context model complexity and at the same time presents the relevant aspects in a form that better suits the needs for the specific application domain. The tool, which relies on the trigger-action paradigm, aims to have a reasonable trade-off between expressivity and ease of use for the average user. We conducted a user study in which participants had to customize a home application with our solution and then filled in a questionnaire. The results of the usability test were overall promising.

To further improve the solution and possibly enlarge the pool of potential end users, we can consider two main directions. One is to revise the domain-specific context model with the aim to make it more intuitive, so as to allow the novices to quickly find the contextual attributes. It is worth pointing out that revising the domain-specific context model does not imply large modifications of the authoring tool but only requires to adjust the context model schema file. The other direction is to consider some participants' issues and indications, such as refining the rule description in natural language and better differentiating trigger and action areas.

Future work will also be dedicated to improving rules debugging to better explain why some rules fail, and further empirical validation by also applying the proposed method and tools in other application domains (for example remote elderly assistance).

9. REFERENCES

S. Ahn and D. Kim. 2006. Proactive context-aware sensor networks. In Proceedings of the Third European conference on Wireless Sensor Networks (EWSN'06), Kay Römer, Holger Karl, and Friedemann Mattern (Eds.). Springer-Verlag, Berlin, Heidelberg, 38-53. DOI=http://dx.doi.org/10.1007/11669463_6

L. Atzori, A. Iera, and G. Morabito. 2010. The Internet of Things: A survey. Computer Networks, Volume 54, Issue 15, 28 October 2010, Pages 2787–2805. doi:10.1016/j.comnet.2010.05.010

C. Buckl, S. Sommer, A. Scholz, A. Knoll, A. Kemper, J. Heuer, and A. Schmitt: Services to the Field: An Approach for Resource Constrained Sensor/Actor Networks. AINA Workshops 2009: 476-481

J. Coutaz, and J. L. Crowley: Learning about End-User Development for Smart Homes by "Eating Our Own Dog Food". In: Proc. of Workshop on End User Development in the Internet of Things Era (EUDITE '15) - CHI '15 EA. Seoul (Korea). April 19, 2015.

J. Coutaz, S. Caffiau, A. Demeure, and J. L. Crowley: Early lessons from the development of SPOK, an end-user development environment for smart homes. UbiComp Adjunct 2014: 895-902

J. Coutaz, and J.L. Crowley: A first person experience with end-user development for smart home. IEEE Pervasive Computing, vol. 15, no 2, May-June 2016: 26:39

O. Davidyuk, I. Sanchez, E. Gilman and J. Riekki: An Overview of Interactive Application Composition Approaches, Open Computer Science. Volume 5, Issue 1,ISSN (Online) 2299-1093, DOI: 10.1515/comp-2015-0007, December 2015.

J. Dax, T. Ludwig, J. Meurer, V. Pipek, M. Stein, and G. Stevens: FRAMES – A Framework for Adaptable Mobile Event-Contingent Self-report Studies. IS-EUD 2015: 141-155.

G. Desolda, C. Ardito, M. Matera, and A. Piccinno, Mashing-up smart things: a meta-design approach. In: Proc. of Workshop on End User Development in the Internet of Things Era (EUDITE '15) - CHI '15 EA. Seoul (Korea). April 19, 2015. pp. 33 - 36..

A. K. Dey, and A. Newberger: Support for context-aware intelligibility and control. CHI 2009: 859-868

A.K. Dey, T. Sohn, S. Streng, and J. Kodama: iCAP: Interactive Prototyping of Context-Aware Applications. Pervasive 2006: 254-271

Z. Drey, and C. Consel: Taxonomy-Driven Prototyping of Home Automation Applications: A Novice-Programmer Visual Language and its Evaluation. Journal of Visual Languages and Computing (JVLC) 23 (2012): 311–326.

G. Fischer, E. Giaccardi, Y. Ye, AG Sutcliffe, N. Mehandjiev, Meta-design: a manifesto for end-user development, Communications of the ACM 47 (9), 33-37, 2004.

G. Ghiani, M.Manca, F.Paternò, Authoring Context-dependent Cross-device User Interfaces based on Trigger/Action Rules, The 14th International Conference on Mobile and Ubiquitous Multimedia (MUM2015), pp. 313-322, ACM Press.

T. Gu, H. K. Pung, and D. Zhang: A service-oriented middleware for building context-aware services. J. Network and Computer Applications 28(1): 1-18 (2005)

D. Guan, W. Yuan, S. Lee, and Y.-K. Lee, Context Selection and Reasoning in Ubiquitous Computing. International Conference on Intelligent Pervasive Computing, IPC 2007, IEEE, 2007

P. Hu, J. Indulska, and R. Robinson: An Autonomic Context Management System for Pervasive Computing. PerCom 2008: 213-223

J. Huang, and M. Cakmak. 2015. Supporting mental model accuracy in trigger-action programming. In Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15). ACM, New York, NY, USA, 215-225. DOI=http://dx.doi.org/10.1145/2750858.2805830

T. Kubitza, S. Thullner, and A. Schmidt: VEII: A Toolkit for Editing Multimedia Content of Interactive Installations On-site. PerDis 2015: 249-250

T. Kubitza, and A. Schmidt: Towards a Toolkit for the Rapid Creation of Smart Environments. IS-EUD 2015: 230-235

B. Y. Lim, and A. K. Dey: Toolkit to support intelligibility in context-aware applications. UbiComp 2010: 13-22

G. Lucci, and F. Paternò: Understanding End-User Development of Context-Dependent Applications in Smartphones. HCSE 2014: 182-198

S. Mayer, A. Tschofen, A.K. Dey, and F. Mattern: User interfaces for smart things--A generative approach with semantic interaction descriptions, ACM Transactions on Computer-Human Interaction (TOCHI) 21 (2), 12, 2014.

B. A. Myers, S. E. Hudson and R. Pausch: Past, Present and Future of User Interface Software Tools, *ACM Transactions on Computer Human Interaction.* March, 2000. **7**(1). pp. 3-28.

C. Perera, S. Aghaee, and A. F. Blackwell: Natural Notation for the Domestic Internet of Things. Proceedings IS-EUD 2015: 25-41, Springer Verlag.

C. Perera, C. H. Liu, S. Jayawardena, and M. Chen: Context-aware Computing in the Internet of Things: A Survey on Internet of Things From Industrial Market Perspective. CoRR abs/1502.00164 (2015)

V. Pipek, and V. Wulf: Infrastructuring: Towards an Integrated Perspective on the Design and Use of Information Technology. Journal of the Association of Information Systems (JAIS), Volume 10, Issue 5,

May 2009, 306-332

D. Salber, A. K. Dey, and G. D. Abowd: The Context Toolkit: Aiding the Development of Context-Enabled Applications. CHI 1999: 434-441

A. G. Sutcliffe, G. Papamargaritis: End-user development by application-domain configuration. Journal of Systems and Software 91: 85-99 (2014)

D. Tetteroo, P. Vreugdenhil, I. Grisel, M. Michielsen, E. Kuppens, D. Vanmulken, and P. Markopoulos: Lessons Learnt from Deploying an End-User Development Platform for Physical Rehabilitation. In Proceedings CHI '15. ACM Press, pp. 4133-4142. DOI=http://dx.doi.org/10.1145/2702123.2702504

B. Ur, E. McManus, M. P. Y. Ho, and M. L. Littman. 2014. Practical trigger-action programming in the smart home. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14). ACM, New York, NY, USA, 803-812. DOI=http://dx.doi.org/10.1145/2556288.2557420

A. H. Van Bunningen, L. Feng, and P. M. G. Apers: Context for Ubiquitous Data Management. UDM 2005: 17-24